Enterprise Collaboration Framework for Managing, Advancing and Unifying the Functionality of Multiple Cloud-Based Services with the Help of a Graph API

Michael Petychakis, Iosif Alvertis, Evmorfia Biliri, Romanos Tsouroplis, Fenareti Lampathaki, and Dimitris Askounis

National Technical University of Athens, School of Electrical and Computer Engineering, Athens, Greece
{mpetyx,alvertisjo,ebiliri,rtsouroplis,flamp,askous}@epu.ntua.gr

Abstract. Nowadays, the proliferation of cloud-based services (CBS) has revolutionized the way people communicate, connect, share and eventually conduct business. Large and small and medium enterprises often adapt to this era by providing their core competence through an API. The present paper aims at describing a solution that manages, advances and unifies the functionality of the various CBS under a common framework that is being developed as a coherent graph API. Since the task of handling the evolution and complexity of the CBS API lifecycle is high, the framework is accompanied by a tool that creates a community of enterprises and developers actively engaged both in the usage of the Generic Graph APIs, but also in the update and improvement of these initial APIs.

Keywords: Web APIs, Cloud-based Services, Generic APIs, Graph API, Social Enterprise, User-centric API Framework.

1 Introduction

Since the early nineties, there has been an explosion of web pages as well as of web services. The era of the web is here with many enterprises trying to adapt and transform their businesses. Each enterprise develops its own approach to expose its services / APIs, and other companies that need to consume such services have to adjust to their requirements. Even though such an approach might have worked in the early days of the Internet, nowadays that there are thousands of available APIs, it is getting harder and harder for enterprises to keep track. For that reason they have dedicated departments only for this purpose. Even though some approaches for common practices and standards have been discussed and proposed during the years, nothing has been yet monetized to its full potential. Some of the most popular approaches are RAML [1], Swagger [2], WADL [3] and API blueprints [4]. One emerging approach worth mentioning is the Hydra [5,6], which tries to build a common description language for web hypermedia APIs utilizing the toolkit extracted by JSON-LD and semantic web.

In this paper, we describe a methodology to gather APIs from various cloud-based services that are important for enterprises, expose a common graph API for all of

L.M. Camarinha-Matos and H. Afsarmanesh (Eds.): PRO-VE 2014, IFIP AICT 434, pp. 153–160, 2014.

them, and provide documentation and a mocking tool to test before actually implementing their applications. Companies are given the opportunity to design, implement and verify their own APIs on top of this unifying API. In order to verify that such a framework is sustainable, we have enhanced it with a community orientation in order for companies and individual developers to directly interact with the framework, and improve it.

The whole methodology was designed and implemented as part of the OPENi EU funded project. Targeting the needs of end-users and application developers, OPENi realizes its vision for user-centric, cloud-connected applications by providing:

- To Mobile Application Users: a personal cloudlet, a single location to store and control their personal data and to realize a novel application experience, by being able to keep their user generated content and data in the cloud and further make it available across different applications on different devices.
- To Developers: an open API framework to build applications that can seamlessly integrate publicly available cloud-based functionality and content and to expand the provided API functionalities, according to their needs, with respect to the Cloudlet owners' privacy.

In section 2 of the present paper, we give an overview of the domain of web APIs and the problems that enterprises face. In section 3, we present the core methodological steps that we followed and in section 4, we describe in detail the whole approach of the framework and how this is beneficial, goingbeyond the current state of the art. In the final chapter, we conclude with a brief overview and the next steps.

2 Problem

There is an increasing discussion around the API Economy, referring to the increasing availability of APIs and making data accessible to third parties [7]. As of 1 May 2014, Programmable Web contained a listing of 11,365 public APIs [8] and is still continuously growing. Building over others' solutions puts many risks and dependencies on a business, with the main problem being that any changes or discontinuities may put more effort in tracking changes in an API, and updating an interface to it. The more dependencies there are with third party APIs, the more complex it becomes for an enterprise to handle. Some APIs are quite stable, like Twitter API reaching only Version 1.1 by 2013 [9], but with major changes in their methods too. Other services have been undergoing major changes for years, like Facebook that only recently has promised that each new API version will be supported for two years [10] and will give developers 90 days of migration; but still the version 1.0 is considered quite unstable to be supported under this plan. Some other APIs have been completely discontinued, like the TESCO API [11]. Under this growing and important business of APIs, the instability and maintenance of interfaces has become a major issue for every developer and more generally for every enterprise that builds interfaces with multiple external data resources as outlined by Jun Li [12]. Thus, a layer of abstraction that will combine multiple API resources, put them under a unique API and govern any API changes can be the solution to this growing uncertainty. A nice similar approach by Tanaka [13], even though it focuses on a

slightly different point, it also outlines the importance of web APIs in the enterprise and why mashups are crucial in the industry.

Another major issue, especially for community-based businesses, is to develop early on an active community, which will provide a platform with unique content of high quality. Existing platforms have created closed silos of content, creating the hazard of a fragmented, closed world as Christian Bizer stated [15], [16]. In some cases also, some platforms have followed blurred lines towards their adoption with copyright issues, like YouTube was built over proprietary video content, Pinterest is based on copyrighted pictures that its users upload or Flipboard was based on content retrieved and filtered by common RSS feed resources. Thus, a common way to access users' data, wherever these are (e.g. their personal storage, Facebook, Instagram, Dropbox etc.), just by authorizing an application to access them, would solve the "cold-start problem" for many digital solutions. At the moment, the central point of reference, content authorization and identity management is Facebook for third party developers. A platform that would allow enterprises to build and reach collaboratively a community of users, by extending it with data structures and business capabilities not imagined before but also without locking this content for specific companies, would solve this problem in the modern world. Such notions have already been discussed in detail by Zhong [14], by explaining the complexity that such systems can carry and by proposing a methodology of how this could be handled.

Last but not least, even if there are best practices and standards about designing APIs, there are no industry-specific APIs guidelines or standards. For example, the Facebook Graph API has evolved to an industry standard in social media applications, something that sooner or later Twitter followed, but there is no best practice in e-shopping solutions. An interoperable platform can become a central point of reference and connectivity for every new API that rolls out, and if built around an open community mentality, it may reduce the costs of maintaining the web of APIs. As part of the state of the art analysis we conducted, we identified that numerous companies have built their whole business model around helping other companies designing and building better APIs. In this paper we go beyond the state of the art, designating a framework that semi-automates all this procedure in a sustainable manner.

3 Methodology

One of the major targets of OPENi is to offer an open API framework that is capable of interoperating with third party services, abstracting the integration challenges to a single open standard. To do that, the following procedure was followed:

1. Extensive analysis of Cloud-based Services with regard to usefulness, popularity, range of functionalities and used technologies. The availability of an API is considered a prerequisite for a service to be considered for integration.
2. Categorization of Cloud-based Services, upon studying similar categorization proposed by developers' reference portals as well as the categorization adopted by the biggest mobile apps marketplaces.
3. A set of Generic APIs (i.e. Activity API, Media API, Products & Services API) was created and specified based on the identified categories. These

Generic APIs will handle interoperability with any cloud-based service, abstracting the integration challenges to a single open standard without losing important service features.

4. Investigation of ways to ensure the sustainability of the Generic APIs, offering the means to enhance them, thus increasing their added value for enterprises that choose to utilize them.

From this last step the need for a new Platform, the API Builder, emerged, envisaged to tackle the following challenges:

- *Easy, semi-automatic integration of changes in CBS*: In today's rapidly evolving landscape, new Cloud Based Services are certain to appear and existing ones will possibly alter their APIs as referenced by Dig and Johnson[17], as well as by Fokaefs [18]. A platform like the API Builder, that aims to offer an abstraction of the various APIs and their partial integration into new unified ones, would lose its advantage without a mechanism to handle such changes. Alterations may appear at any level of the Cloud Based Services' ecosystem, including minor/major changes to or discontinuation of existing services, extension with new methods and appearance of new CBS.
- *Documentation*: When incorporating services provided by multiple platforms, the need to study and understand all the different APIs can be a very frustrating and time-consuming task. Therefore, the existence of an inclusive documentation for all the services in the same consistent format was considered to be of great value for the developers and enterprises.
- *Flexibility*: Not all enterprises need to offer the same services and thus not all are in need of the same API. Even without considering the possible changes in the underlying APIs, the need to extend, customize and build on top of it is evident and should therefore be accounted for.
- *Avoid duplicate effort*: The size of the market targeting the Cloud Based Services has led to the appearance of numerous products and services that utilize these 3rd party APIs, many of which required the development of similar middle components. Developers and thus enterprises would greatly benefit from a platform that allows the designation and re-use of already designed and/or implemented components, discouraging investing effort to duplicate existing work.
- *Lack of user pool*: Despite the boom of applications exploiting the CBS, many fail to establish their role in the market. The API Builder, as part of a large platform with an existing user pool, usage statistics and clear predefined privacy settings can offer new opportunities to the application developers.

Having a different API for every object would require more clients to be implemented, more calls within the platform and would ultimately increase the complexity of using as well as sustaining such a platform. To this end, graph modeling was selected as the more suitable and intuitive approach in order to properly handle the enormous amount of information that needs to be stored and also the need to dynamically extend and update the underlying schema. That is why all the OPENi created meta-model follow the Linked Data paradigm. The schema.org vocabulary has been actively indexed in a way that every API and all of its objects are mapped by a one to one relationship to a vocabulary entry.

4 Approach

Figure 1 High Level Architecture depicts the API Builder's role within the OPENi ecosystem.

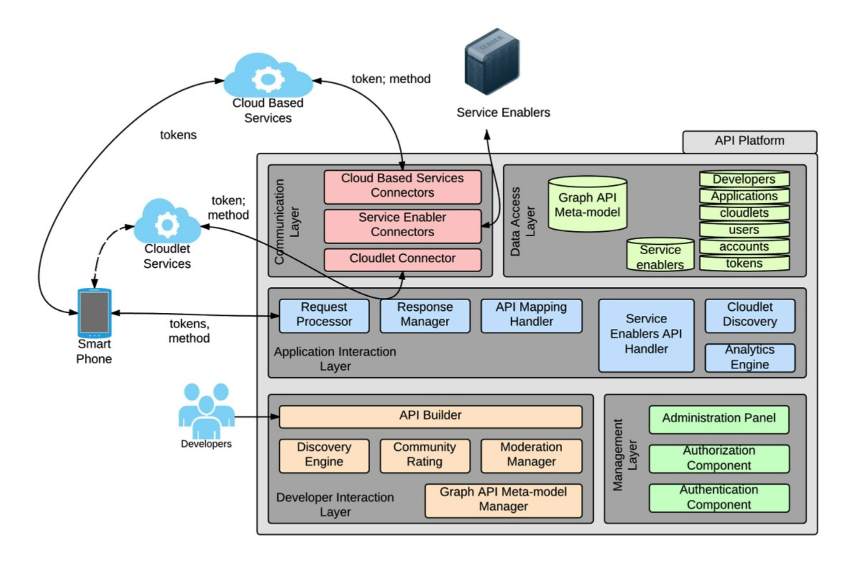


Fig. 1. High Level Architecture

As already explained in the Methodology section, it is important to sustain the Platform and make it viable. Changes are not always easy to trace automatically and for that reason the API Builder is designed as a collaborative tool for developing APIs.

The OPENi APIs provide a number of objects and methods. Changes can be handled at object, method and CBS levels, in the following ways:

- Object Creation: Everyone participating in the Builder can create new object schemas. In order to provide full flexibility, every OPENi object, either part of the initial corpus or created inside the API Builder can be used as part of another object.
- Method creation-customization: Each participating enterprise can add, remove and rename fields from existing methods as needed or even combine methods, essentially creating new ones.
- Semi-automatic integration of changes in the CBS APIs: Minor changes in a 3rd party API could be handled through the method creation and customization functionalities. For large-scale API changes, a separate mechanism should be made available to enterprises that includes code submission, review and integration phases. Similar effort has been undertaken by Taheriyan [19], trying to create a semi-automatic way to map web APIs.
- Support for integration of a new CBS: As the Cloud Based Services are part of a rapidly changing market, new services may emerge. Hiding the unnecessary complexity from the enterprises is a key feature of the API Builder, but the

integration of a new CBS is not expected to be provided as a fully automated process by the API Builder. To integrate the API of a previously unsupported Cloud Based Service, separate changes will be required including code submission to the OPENi repository, additions to the documentation and extension of the underlying graph schema. Excluding any of these steps will leave the OPENi platform in an inconsistent state. To avoid that, the API Builder should provide developers with an intuitive interface that guides them through the completion of the necessary steps, enforcing the correct procedure flow, while in the background checking if the proposed changes should be made in the underlying model.

To achieve efficient exploitation of the provided functionalities which are aggregated from the OPENi cloudlet platform and from 3rd party APIs, enterprise developers are in need of sufficient guidance. That is why extensive documentation is provided through the Builder's UI for all the exposed services.

The Builder greatly depends on its community of developers to actively participate and take advantage of these mechanisms in order to keep the API framework up-to-date, functional and intuitive. As pointed out by Pankowska in [20], the authority of a collective community increases sustainability. For that reason, the API Builder is designed to offer a number of social networking functionalities.

It should be noted that although sharing objects and methods within the community is strongly encouraged, an enterprise might choose not to disclose its work. However, the envisaged collaboration network is expected to help developers deliver the required product in less time which ultimately translates into profit for the enterprises. The collaboration mechanisms are adopted for two more reasons.

First, any API that exists in the platform is also exposed for validation as well as feedback to the community. If a useful component is implemented, it is expected to gain high score and thus be more easily detected and re-used, which can serve as a first level of duplicate effort avoidance. To further empower this methodology, a recommendation system is used for suggesting objects and APIs to a developer interested in building a new application on top of the OPENi ecosystem [Figure 1].

The second reason is related to the smoother adoption of applications that interact with the API Platform. Since developers reuse the schemas that others have used, some data conforming to the first schema probably already exist in some users' storage (cloudlet). That way, the new application can directly utilize this data, as far as this is allowed by the user's privacy settings. Within the OPENi Framework, clean privacy rules have been of primary importance and users' data are stored and treated with respect to the desired level of privacy. It would thus be easier for users to trust these applications, as they are guaranteed to follow well-known practices concerning data privacy.

This can prove to be essential for the community of applications, because especially in the early stages a large community of users cannot be ensured. With the proposed procedure this is not important since we facilitate reusing preexisting information and do not allow for silos of data within the framework.

Enterprises could also benefit from the provided API usage statistics in order to gain insights about the utility of the implemented APIs but also their popularity which could disclose new trends.

5 Conclusions

A paradigm shift in enterprise-centric design of business applications is emerging based on the proliferation of APIs that play a pivotal role in a thriving API ecosystem by unlocking latent value in data and information of cloud-based services. With the purpose of supporting easy integration of a broad spectrum of existing cloud-based functionality in a platform-independent way, a framework supported by a tool has been shaped. This framework takes into account the various usages that companies may find for interacting with the CBS and the needs of those when coming to collecting information for analytics or in general collecting information. Those issues when coming to providing a generic method for interacting with existing APIs as well as handling the data privacy issues and all the complexity that comes from interconnecting privacy restrictions at different layers and portions of data, have been delivered in this framework. A step further is also taken by providing a community of developers collaborating with the enterprises in creating APIs and dealing with the complications.

The OPENi platform provides an added-value approach that aims to tackle the diversity of the above-enumerated challenges. Its architecture and implementation are both motivated by the research challenges addressed in this paper. It is a place where enterprises and developers can confidently cooperate into constructing better APIs by improving the developing API lifecycle and reducing the privacy complexity issues as well as the tracking changes effort.

As next steps, we are going to have a deeper integration with hypermedia API description standards, enhance interoperability with enterprise systems and validate our approach in the domains of advertising, shopping and personal data management. As an ultimate goal, it would be favorable to have each API provider to have their API described directly within this framework and improve it with any new changes.

Acknowledgments. This work has been created in the context of the EU-funded project OPENi (Open-Source, Web-Based, Framework for Integrating Applications with Social Media Services and Personal Cloudlets), Contract No: FP7-ICT-317883.

References

1. RAML, http://raml.org/ (accessed: April 04, 2014)
2. Swagger: A simple, open standard for describing REST APIs with JSON, Reverb Technologies (2013), https://developers.helloreverb.com/swagger/ (accessed: April 04, 2014)
3. Hadley, M.J.: Web Application Description Language, W3C Member Submission (2009), http://www.w3.org/Submission/wadl/ (accessed: April 04, 2014)
4. API Blueprint, http://apiblueprint.org/ (accessed: April 04, 2014)
5. Lanthaler, M.: Creating 3rd generation web APIs with hydra. In: Proceedings of the 22nd International Conference on World Wide Web Companion, pp. 35–38. International World Wide Web Conferences Steering Committee (2013)

6. Lanthaler, M.: Leveraging Linked Data to Build Hypermedia-Driven Web APIs. In: REST: Advanced Research Topics and Practical Applications, pp. 107–123. Springer, New York (2014)
7. A Survey of the API Economy. Israel Gat, Giancarlo Succi. Agile Product & Project Management. Executive Update 14(6)
8. ProgrammableWeb API Directory, http://www.programmableweb.com/apis (accessed: April 04, 2014)
9. History of the REST & Search API. Twitter, https://dev.twitter.com/docs/history-rest-search-api (accessed: May 03, 2014)
10. Platform Versioning. Facebook Developers (May 03, 2014), https://developers.facebook.com/docs/apps/versions (accessed: May 03, 2014)
11. Tesco API discontinued. ProgrammableWeb Directory, http://www.programmableweb.com/api/tesco (accessed: May 03, 2014)
12. Li, J., Xiong, Y., Liu, X., Zhang, L.: How Does Web Service API Evolution Affect Clients? In: IEEE 20th International Conference on Web Services (ICWS), June 28-July 3, pp. 300–307 (2013)
13. Tanaka, M., Kume, T., Matsuo, A.: Web API Creation for Enterprise Mashup. In: 2011 IEEE World Congress on Services (SERVICES), July 4–9, pp. 319–326 (2011)
14. Zhong, J., Bertok, P., Tari, Z.: Security, Privacy and Interoperability in Heterogeneous Systems. In: Camarinha-Matos, L.M., Boucher, X., Afsarmanesh, H. (eds.) PRO-VE 2010. IFIP AICT, vol. 336, pp. 713–721. Springer, Heidelberg (2010)
15. Bizer, C., Heath, T., Idehen, K., Berners-Lee, T.: Linked data on the web (LDOW2008). In: Proceedings of the 17th International Conference on World Wide Web (WWW 2008), pp. 1265–1266. ACM, New York (2008), http://doi.acm.org/10.1145/1367497.1367760, doi:10.1145/1367497.1367760
16. Bizer, C., Heath, T., Berners-Lee, T.: Linked data-the story so far. International Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)
17. Dig, D., Johnson, R.: The role of refactorings in API evolution. In: Proceedings of the 21st IEEE International Conference on Software Maintenance, ICSM 2005, September 26-29, pp. 389–398 (2005)
18. Fokaefs, M., Mikhaiel, R., Tsantalis, N., Stroulia, E., Lau, A.: An Empirical Study on Web Service Evolution. In: Proceedings of International Conference on Web Services (ICWS 2011), pp. 49–56 (2011)
19. Taheriyan, M., et al.: Semi-Automatically Modeling Web APIs to Create Linked APIs. In: Proceedings of the First Linked APIs Workshop at the Ninth Extended Semantic Web Conference (2012)
20. Pankowska, M.: Sustainability of Virtual Collaborative Networks. In: Camarinha-Matos, L.M., Picard, W. (eds.) Pervasive Collaborative Networks. IFIP, vol. 283, pp. 85–92. Springer, Boston (2008)